Assert in Prolog

Mattox Beckman

beckman@iit.edu

Illinois Institute of Technology

Objectives

You can often tell what the language designers thought about their language by the libraries that are included with it. Many of Prolog's involve the analysis of structres. In this lecture we will go over some of the builtin predicates of Prolog.

- Know how to use `call` and `assert`
- Know how to use `findall` and `checklist`.

Two useful list predicates

- `findall(X,T,Y)` finds all values of `X` that make `T` true, and puts them into `Y`.
- `checklist(P,Y)` is true if predicate `P` is true for all values in list `Y`.

```
?- findall(X,possible(X),Xs).
X = _G306
Xs = [anna, beth, cindy, david, ernest, frank, gloria
?- checklist(student,[anna,beth]).
Yes
24 ?- checklist(student,[anna,harry]).
No
```


Pairs

- The term `socrates` is a pattern. But patterns can have structure....

```
pair((X,Y)).
key((X,Y),X).
value((X,Y),Y).
assoc(X,Y,[H|T]) :- key(H,X), value(H,Y);
                    assoc(X,Y,T).
?- assoc(2,X,[(3,hi),(4,there),(2,guys)]).
X = guys
?- assoc(X,there,[(3,hi),(4,there),(2,guys)]).
X = 4
```

Here we use a pattern called `bst`, which is a *functor*.

```
1  find(X,bst(X,_,_)).
2  find(X,bst(Y,A,_)) :- X < Y, find(X,A).
3  find(X,bst(Y,_,B)) :- X >= Y, find(X,B).
4  ?- find(3,bst(4,bst(5,null,null),bst(3,null,null))).
5  No
6  ?- find(3,bst(4,bst(3,null,null),bst(5,null,null))).
7  Yes
8  ?- find(X,bst(4,bst(3,null,null),bst(5,null,null))).
9  X = 4 ;
10 ERROR: Arguments are not sufficiently instantiated
11    Exception: (6) find(_G474, bst(4, bst(3, null, nul
```



```
1  bstgt(X,null).
2  bstgt(X,bst(A,_,_)) :- X > A.
3  bstle(X,null).
4  bstle(X,bst(A,_,_)) :- X =< A.
5  isbst(null).
6  isbst(bst(X,A,B)) :- bstgt(X,A), isbst(A),
7                       bstle(X,B), isbst(B).
8  ?- isbst(bst(4,bst(3,null,null),bst(5,null,null))).
9  Yes
10 ?- isbst(bst(4,bst(5,null,null),bst(3,null,null))).
11 No
```


One power that Prolog programs have is the ability to examine and modify themselves.

- Used for AI — real learning requires the ability to “examine yourself”.
- Prolog structures and prolog programs have the same form.
  - Assembly language: bit patterns
  - Scheme and Lisp: lists
  - Prolog: structures

```
1  likes(john,mary).
2  ?- isbst(bst(5,null,null)).
```

Functors: `likes`,`isbst`, and `bst`.

We have predicates that will determine the type of a term.

```
1  ?- atom(3).
2  No
3  ?- atom(hi).
4  Yes
5  ?- atomic(3).
6  Yes
7  ?- integer(3).
8  Yes
9  ?- integer(f).
10 No
```

```
1  ?- X = 20, integer(X).
2  X = 20
3  Yes
4  ?- var(X).
5  Yes
6  ?- X = 20, var(X).
7  No
```

Name

The `name` predicate turns a term into a string (and back).

```
?- name(foo,X).
X = [102, 111, 111]
?- name(X,"foo").
X = foo
chop(X,Y) :- name(X,[_|S]), name(Y,S).
?- chop(asymmetric,X).
X = symmetric
```

This will be very useful for natural language processing.

Look what you've done!

- The `listing` predicate will print out the definitions we have so far.

```
?- listing(mortal).

mortal(A) :-
        human(A).

Yes
```


Accessing Parts of Functors

- `functor(T,F,N)` — `F` will contain the name of the functor, `N` will contain the number of arguments.
- `arg(N,T,A)` — `A` will be argument number `N` of `T`

```
-? functor(isbst(5,null,null),F,N).
F = isbst
N = 3
-? arg(1,isbst(5,null,null),A).
A = 5
```


The =.. operator

Another way to deconstruct terms is with "`=..`".

```
?- bst(5,null,null) =.. L.
L = [bst, 5, null, null] ;
?- L =.. [likes,john,X].
L = likes(john, _G276)
X = _G276
(mortal(X) :- human(X)) =.. L.
X = _G324
L = [ (:-), mortal(_G324), human(_G324)]
```

Note that `:-` is a functor!

Database modification

- `assert` allows you to modify things while prolog is running.
- This only works for “dynamic” procedures, though.
- `retract` allows you to undo an assertion.

```
?- assert(prime(2)).
?- assert(prime(3)).
?- assert(prime(5)).
?- assert(prime(7)).
?- prime(3).
Yes
?- retract(prime(3)).
?- prime(3).
No
```


Making things dynamic

```
?- dynamic likes/2.
?- likes(john,mary).
No
?- assert(likes(X,Y) :- likes(Y,X)).
?- assert(likes(john,mary)).
?- likes(mary,X).
ERROR: Out of local stack
?- retract(likes(john,mary)).
Yes
?- asserta(likes(john,mary)).
Yes
?- likes(mary,X).
X = john
```


Executing Code

- The `call` predicate will execute its argument.
- Note that implications are asserted, not called.

```
ask_about(X,Y) :- Q =.. [Y,X], call(Q).
?- ask_about(socrates,mortal).
Yes
?- call(funny(X) :- human(X)).  See, I told you....
ERROR: Undefined procedure: (:-)/2
?- assert(funny(X) :- human(X)).
X = _G324
Yes
?- funny(X).
X = socrates
X = muller
```


Example: answer

Now you can use prolog to keep track of students’ questions.

```
answer(X) :- question(X,Q), !, write(Q),
             retract(question(X,Q)), call(Q).
?- assert(question(jonny,mortal(muller))).
?- assert(question(jonny,mortal(socrates))).
?- answer(jonny).
mortal(muller)
Yes
?- answer(jonny).
mortal(socrates)
Yes
?- answer(jonny).
No
```

Problems

1. Write a function "`says`" that takes two arguments. The first is the name of a person making the implication. The second is a prolog expression. Record the claim, and then tell the prolog.
2. Next, suppose we can find out later that some people aren't reliable, and we should no longer believe anything they say. Write a function "`disbelieve`" that takes the name of a person and retracts everything they said before.

```
1 ?- says(frank,likes(john,mary)).
2 ?- likes(john, mary).
3 Yes
4 ?- disbelieve(frank).
5 ?- likes(john, mary).
6 No
```


Answers

```
1 says(P,X) :- assert(claims(P,X)), assert(X).
2 disbelieve(X) :- findall(Y,claims(X,Y),YL),
3                  checkall(retract,YL).
```